Chapter 3

Distributed File System

Introduction

- In single processor system, the job of a file system is to store programs and data and then when called it makes them available to the user.
- A file is a collection of data with a user view (file structure) and a physical view (blocks).
- A directory is a file that provides a mapping from text names to internal file identifiers.
- File systems implement file management following tasks:
  - Naming and locating a file
  - Accessing a file – create, delete, open, close, read, write, append, truncate
  - Physical allocation of a file.
  - Security and protection of a file.

Introduction

- We must understand the difference between file service and file server.
- File service is the specification of what the file system offers to its clients.
- It describes the primitives available, what parameters they take and what actions they can do.
- From the user point of view, the file service define what services they can have on their files (provided as file system's interface) without knowing how they are implemented. (see the windows services for the files)
- A file server on the other hand is a process that runs on a machine and helps implement the file service.

- The same tasks are provided by the distributed file system with other added tasks (see them later).
- A distributed file system (DFS) is a file system with distributed storage and distributed users.
- Files may be located remotely on servers, and accessed by multiple clients.
  - E.g., SUN NFS and AFS
- A distributed system may have one or more file servers.
- However, the user should not know if there is a single or multiple servers. Ideally, it should look like single-processor file system
- For example, a system may have two file servers one for Unix and the other for Dos with each process runs what is most appropriate for it.
- Thus, the same client can have two opened windows one in which Unix is running and the other with Dos is running with no conflict.

- DFS provides transparency of location, access, and migration of files.
- DFS systems use cache replicas for efficiency and fault tolerance

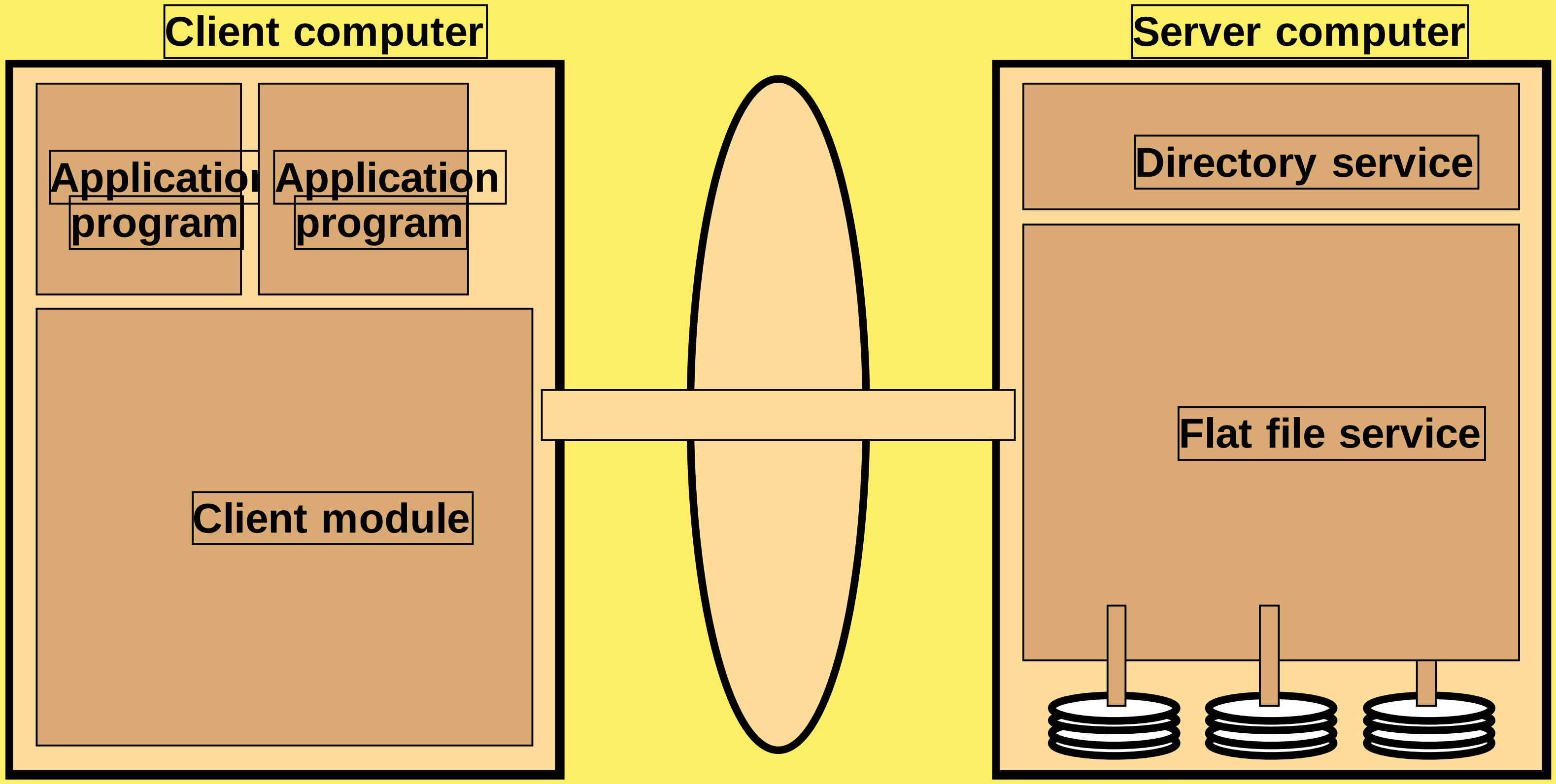
Distributed File System (DFS) Requirements

- Transparency - server-side changes should be invisible to the client-side.
  - *Access transparency*: A single set of operations is provided for access to local/remote files.
  - *Location Transparency*: All client processes see a uniform file name space.
  - *Migration Transparency*: When files are moved from one server to another, users should not see it
  - *Performance Transparency*
  - *Scaling Transparency*

- File Replication
  - A file may be represented by several copies for service efficiency and fault tolerance.
- Concurrent File Updates
  - Changes to a file by one client should not interfere with the operation of other clients simultaneously accessing the same file.
- Concurrent File Updates
  - One-copy update semantics: the file contents seen by all of the processes accessing or updating a given file are those they would see if only a single copy of the file existed.
- Fault Tolerance
  - At most once invocation semantics.
  - At least once semantics. OK for a server protocol designed for idempotent operations (i.e., duplicated requests do not result in invalid updates to files)

- Security
  - Access Control list = per object, list of allowed users and access allowed to each
  - Capability list = per user, list of objects allowed to access and type of access allowed (could be different for each (user,obj))
  - User Authentication: need to authenticate requesting clients so that access control at the server is based on correct user identifiers.
- Efficiency
  - Whole file v.s. block transfer

File Service Architecture



Basic File Service architecture

- It divides the file service into three components:

1- Flat file service

- Implements the operations on the file: create, delete, read, write, get attribute, set attribute and access control operations. It uses UFID to refer to files in all requests for flat file service operations.

2- Directory service:

- It is itself a client of (i.e., uses) flat file service. It provides mapping between text names and UIFD for each file for flat file service.
- It also creates and updates directories (hierarchical file structures)

3- Client service:

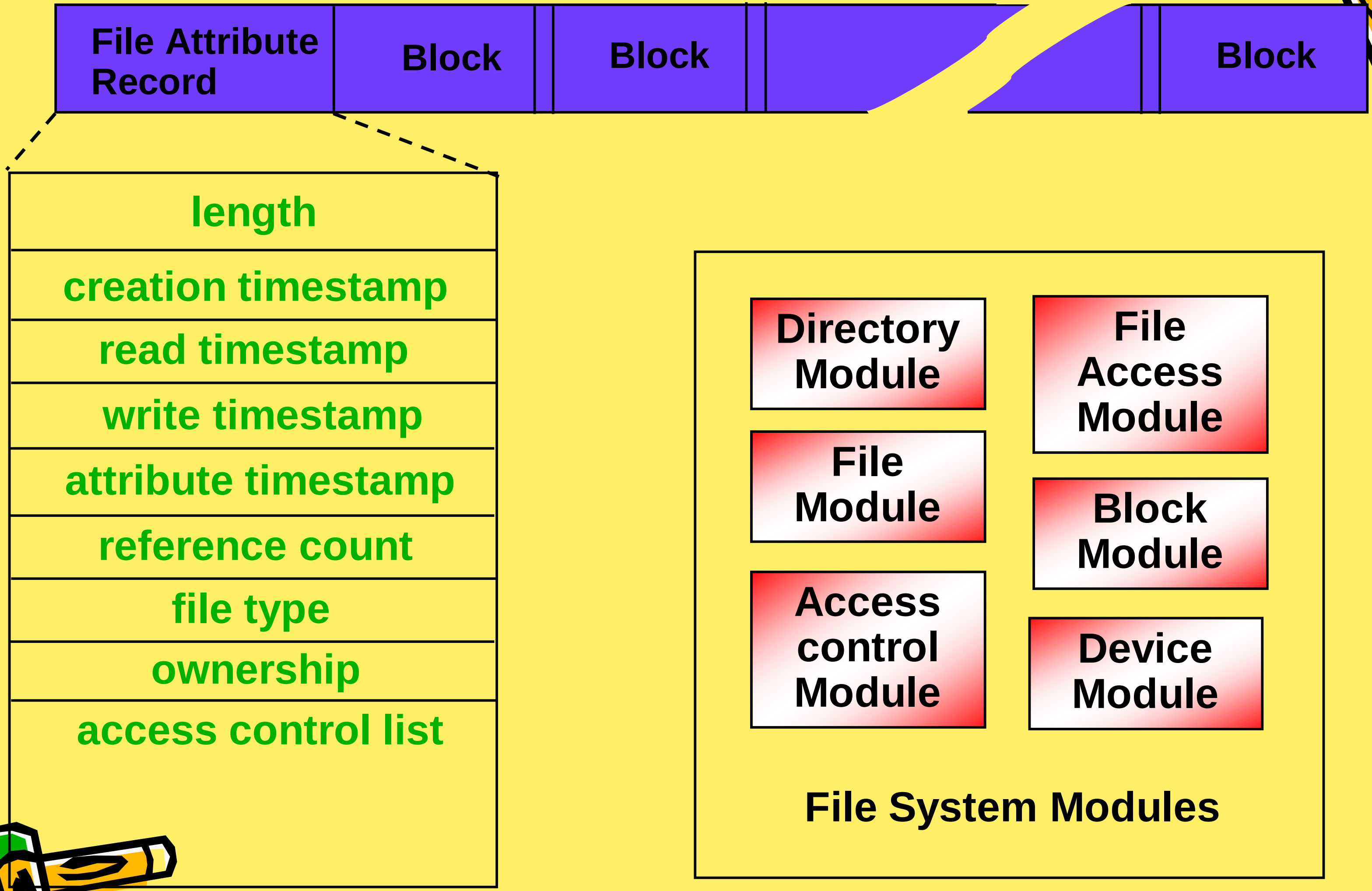
- A client of directory and flat file services
- Runs in each client's computer, integrating and expanding flat file and directory services to provide a unified application programming interface (API)
- The clients holds information about the locations of the flat file server and directory server processes.

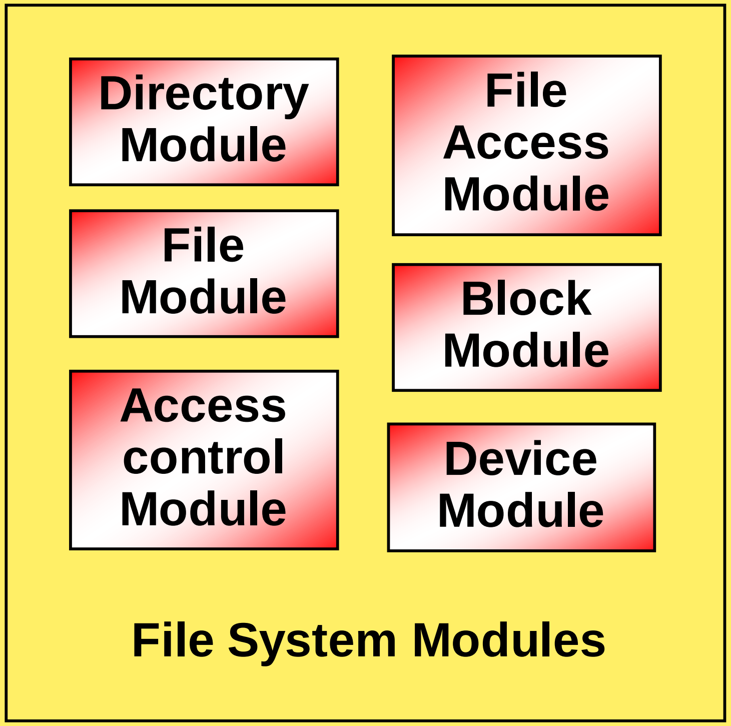
The file service interface

- A DF System typically has two components:
  - True (flat) file service
  - directory service
- The first is concerned with the operations on individual files such as reading, writing, and appending
- The second is concerned with creating and managing directories, adding deleting files from directories
- A file is simply defined to be a uninteruppted sequence of bytes.
- A file can have attributes which are information about the file such as its owner, size, creation date and access permission.
- These attributes are not a part of the file .

- The file service usually provides primitives to read and write some of these attributes.
- For example, you can change the access permission but you can not change its size.
- Other system can provide other primitive and can even accept user-defined primitives.
- Even more, some distributed operating systems can put restrictions on some services offered to a file.
- For example, a file once created it can only be read but can not be modified. (called immutable file).
- Such files are easily cached and replicated without problems or concerns.
- Access control list contains for every file a list of users that can use it and the type of operations that can be performed.

File Attributes & System Modules





File System Modules

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

(This is for Single host File system.

DFS may require additional components.)

Layered architecture: each layer depends only on the layers below it.

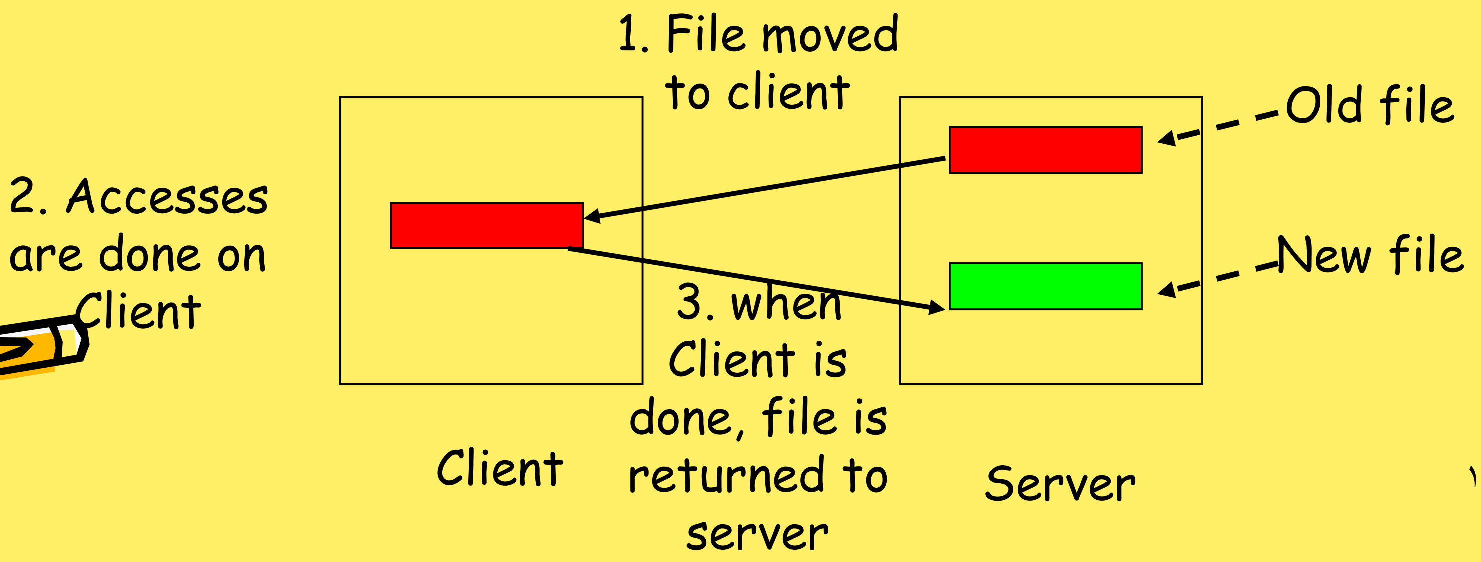
Access Control

- In UNIX, the user's access rights are checked against the access mode requested in the open call and the file is opened only if the user has the necessary rights.
- In DFS, a user identity has to be passed with requests
- A server first authenticates the user.
  - An access check is made whenever a file name is converted to a UFID (unique file id), and the results are encoded in the form of a capability which is returned to the client for future access.
  - Capability = per user, list of objects allowed to access and type of access allowed (could be broken up per (user,obj))
  - A user identity is submitted with every client request, and an access check is performed for every file operation.

File services models

1- Upload/download model (*Cached System*)

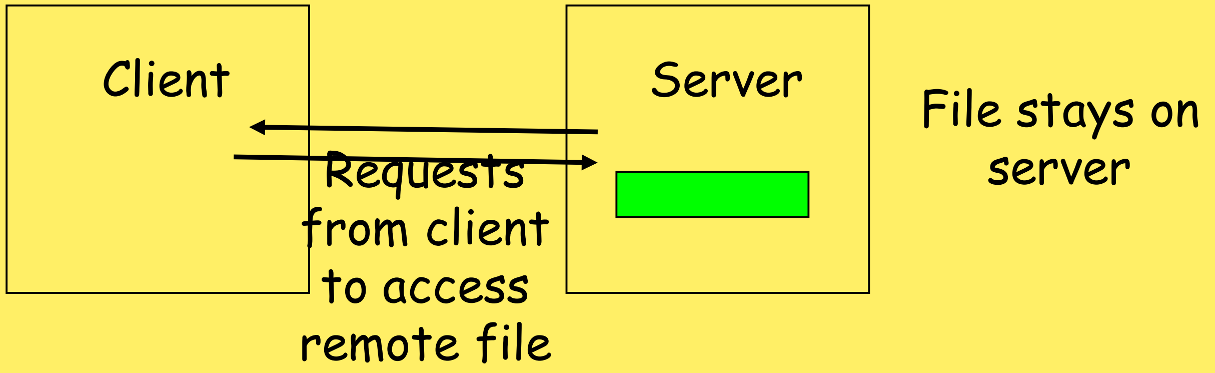
- The file service provides only two major operations: read file / write file.
- The read file operation transfers the entire file from the server to the client.
- Files identified with one master copy residing at the server machine, but copies of (parts of) the file are scattered in different caches
- Reduce network traffic by retaining recently accessed disk blocks in a cache, so that repeated accesses to the same information can be handled locally


1. File moved to client
Old file
New file
2. Accesses are done on Client
3. when Client is done, file is returned to server
Client
Server

- The write file operation transfer an entire file in the other direction from the client to the server.
- Advantages: Many “remote” accesses handled efficiently by the local cache
  - Most served as fast as local ones.
  - Servers contacted only occasionally
    - Reduces server load and network traffic.
    - Enhances potential for scalability.
  - Reduces total network overhead
- It is simple as they will require local usage of files and thus the file service interface is simple.
- The disadvantages: when the client’s storage is small to hold the file and that even if the client uses a part of the file, the whole file is transferred wasting time and network BW.
  - Cache-consistency problem – keeping the cached copies consistent with the master file
    - Could be called network virtual memory

2- Remote access model (*Remote Service*)

- All file actions implemented by server.
- The file service provides many operations for opening and closing files, reading and writing parts of file, moving inside the file seeking for something, examining and changing the file attributes, etc...
- The file system runs on the servers not on the clients
  - RPC (Remote Procedure Calls) functions
  - Use for small memory diskless machines
  - Particularly applicable if large amount of write activity



Cache Location – Disk vs. Main Memory

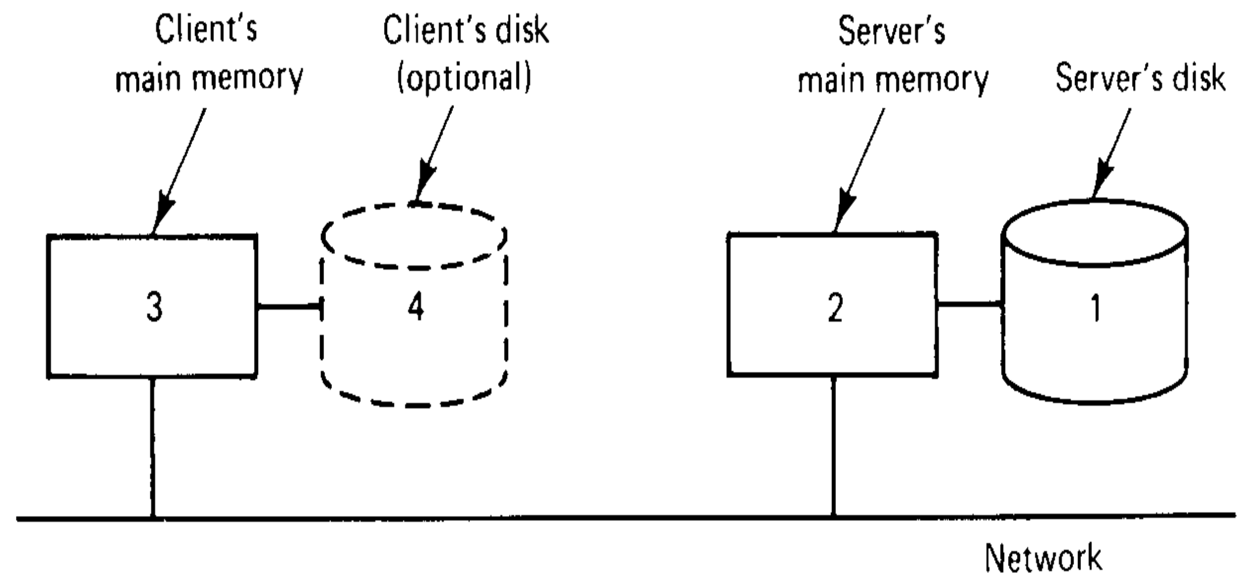


Fig. 5-9. Four places to store files or parts of files.

Cache Location – Disk vs. Main Memory

- Advantages of disk caches
  - More reliable
  - Cached data kept on disk are still there during recovery and don’t need to be fetched again

- Advantages of main-memory caches:
  - Permit workstations to be diskless
  - Data can be accessed more quickly
  - Performance speedup in bigger memories
  - Server caches (used to speed up disk I/O) are in main memory regardless of where user caches are located; using main-memory caches on the user machine permits a single caching mechanism for servers and users

Consistency

- **Is locally cached copy of the data consistent with the master copy?**

- Client-initiated approach
  - **Client initiates a validity check**
  - **Server checks whether the local data are consistent with the master copy**

- Server-initiated approach
  - **Server records, for each client, the (parts of) files it caches**
  - **When server detects a potential inconsistency, it must react**

Cache Update Policy

- Write-through – **write data through to disk as soon as they are placed on any cache. When another client reads the file, he will get the most update from the server**
  - **Reliable, but poor performance**
- Delayed-write – **modifications written to the cache and then written through to the server later**
  - **Write accesses complete quickly; some data may be overwritten before they are written back, and so need never be written at all**
  - **Poor reliability; unwritten data will be lost whenever a user machine crashes**
- **Variation 1 – scan cache at regular intervals and flush blocks that have been modified since the last scan**
- **Variation 2 –** write-on-close, **writes data back to the server when the file is closed**
    - **Best for files that are open for long periods and frequently modified**

Comparing Caching and Remote Service

- In caching, many remote accesses handled efficiently by the local cache; most remote accesses will be served as fast as local ones
- Servers are contracted only occasionally in caching (rather than for each access)
  - Reduces server load and network traffic
  - Enhances potential for scalability
- Remote server method handles every remote access across the network; penalty in network traffic, server load, and performance
- Total network overhead in transmitting big chunks of data (caching) is lower than a series of responses to specific requests (remote-service)

Caching and Remote Service (Cont.)

- Caching is superior in access patterns with infrequent writes
  - With frequent writes, substantial overhead incurred to overcome cache-consistency problem
- Benefit from caching when execution carried out on machines with either local disks or large main memories
- Remote access on diskless, small-memory-capacity machines should be done through remote-service method
- In caching, the lower intermachine interface is different from the upper user interface
- In remote-service, the intermachine interface mirrors the local user-file-system interface

The directory server interface

- The directory service provides operations for:
  - Creating and deleting directories
  - Naming and renaming of files
  - Moving files from one directory to another
- These services are performed on the local or remote file.
- Directory server uses some alphabet and syntax for the file and directory names.
- Some systems divide names into two parts such as prog.c or amany.doc (second part specifies the type)
- Others have an explicit attribute for this purpose instead of tracking the type onto the name.
- All distributed systems provide the possibility of a directory to contain directory.
- When listed, a directory shows files in it only.

- Directories are usually arranged as tree often called hierarchical file system.
- In some other powerful systems, it is possible to create link (or pointer) to an arbitrary directory which makes it possible to make a graph called directory graph.
- A problem occur when a part of the figure is removed specially in distributed system.

Naming and transparency

- Naming – mapping between logical and physical objects
- A transparent DFS hides the location where in the network the file is stored.
- This implies two types of transparency: location transparency and location independence.
- Location transparency – file name does not reveal the file's physical storage location. It is not easy to achieve.
- A file name like server1/distributed/amany.doc tells every thing about file amany.doc as it is stored on server1 but it does not say where server1 is located

Naming and transparency

- Thus server1 can be moved anywhere on the network without any problem to the file name. Thus this system has location transparency.
- But when the file is moved to server2 instead, a problem occurs as in programs where has the file name and place will not work.
- The system which works even the file place is changed has location independency.

- Location transparency – file name does not reveal the file's physical storage location

- Location independence – file name does not need to be changed when the file's physical storage location changes

Naming Schemes — Three Main Approaches

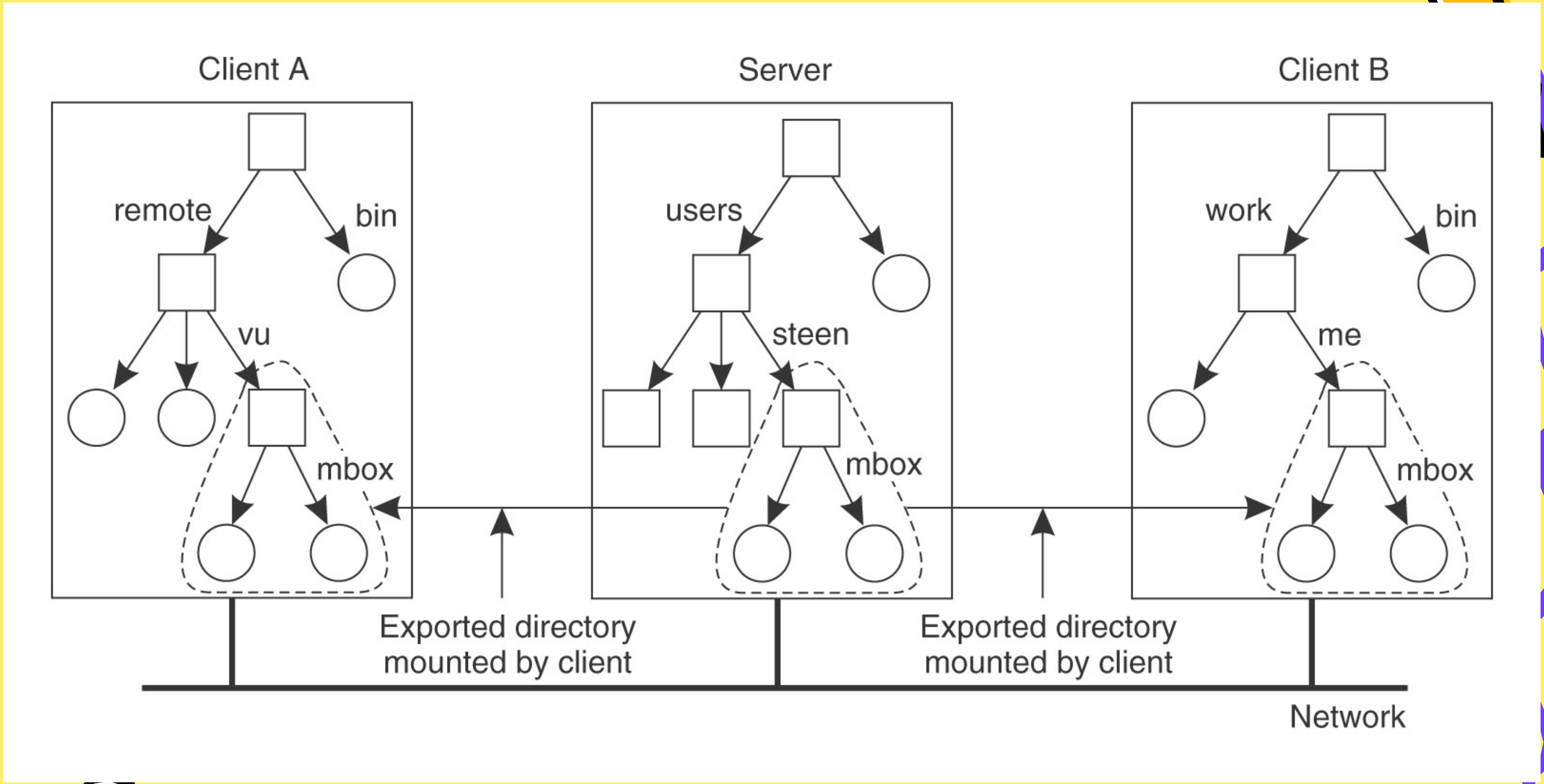
1- Files are named by a combination of their host name and local name; guarantees a unique systemwide name such as /machnie1/path

2- Attach (*Mount*) remote directories to local directories, giving the appearance of a coherent directory tree; only previously mounted remote directories can be accessed transparently

3- A single name space that looks the same on all machines that contains all files in the system

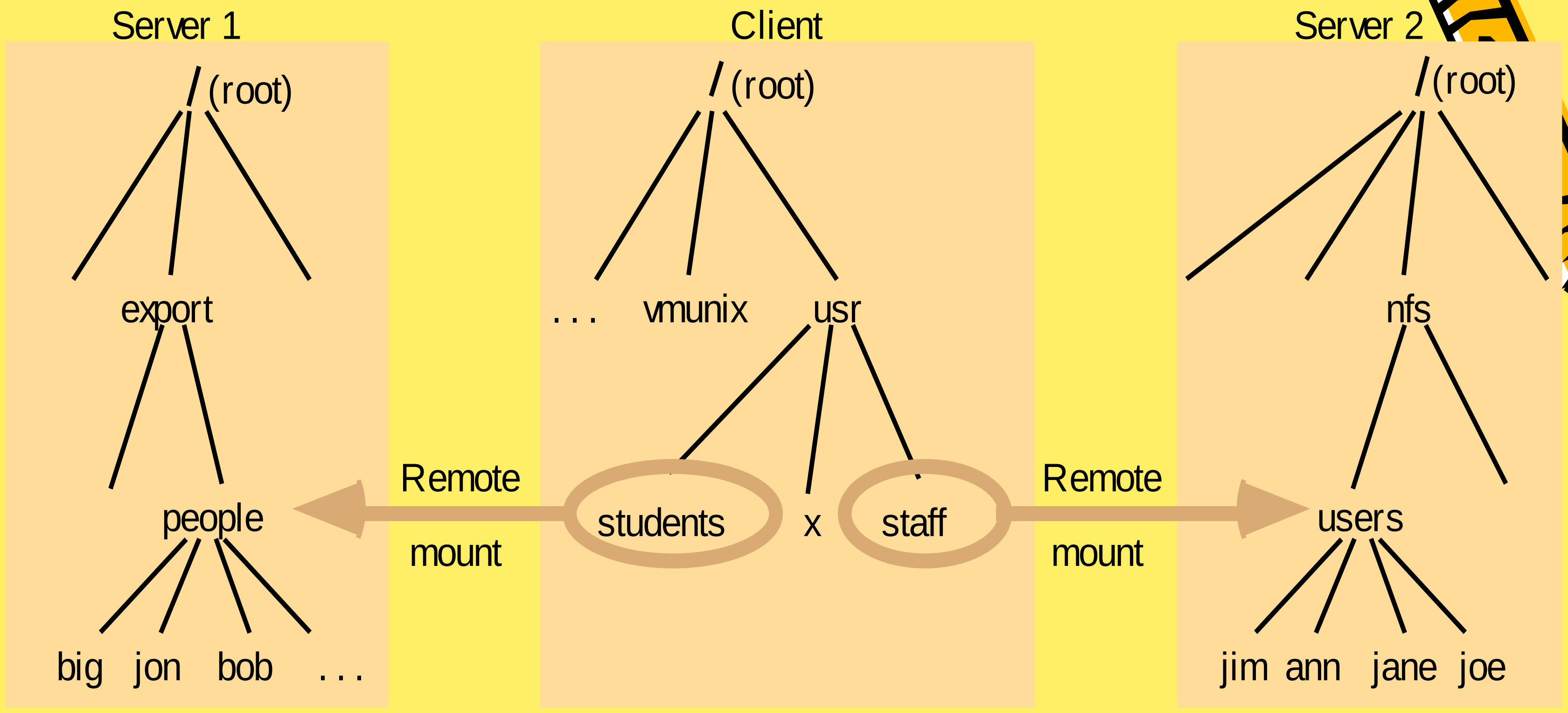
- Total integration of the component file systems
  - If a server is unavailable, some arbitrary set of directories on different machines also becomes unavailable

Mounting Remote Directories (NFS)


Client A
Server
Client B
remote
bin
vu
mbox
users
steen
mbox
work
bin
me
mbox
Exported directory mounted by client
Exported directory mounted by client
Network

Mounting Remote Directories (continued)

- Note:– *names of files are not unique*
  - As represented by *path names*
- E.g.,
  - Server sees : */users/steen/mbox*
  - Client A sees: */remote/vu/mbox*
  - Client B sees: */work/me/mbox*
- Consequence:– Cannot pass file "names" around haphazardly



Note: The filesystem mounted at *\/usr/students* in the client is actually the sub-tree located at *\/export/people* in Server 1;
the file system mounted at *\/usr/staff* in the client is actually the sub-tree located at *\/nfs/users* in Server 2.

State of Service and Client

- How much *state* does the service maintain about its clients?????
- *Stateless*
- *Stateful*

Stateless File Server

- When a client sends a request to a server, the server carries the request, sends the reply, and then remove from its internal tables all information about the request. Between requests, there is no client-specific information kept on the server.

- This means it avoids state information by making each request self-contained (i.e. Each request identifies the full file name and position (offset) in the file) to help the server to do its work

- No need to establish and terminate a connection by open and close operations

- Poor support for locking or synchronization among concurrent accesses

Stateful File Service

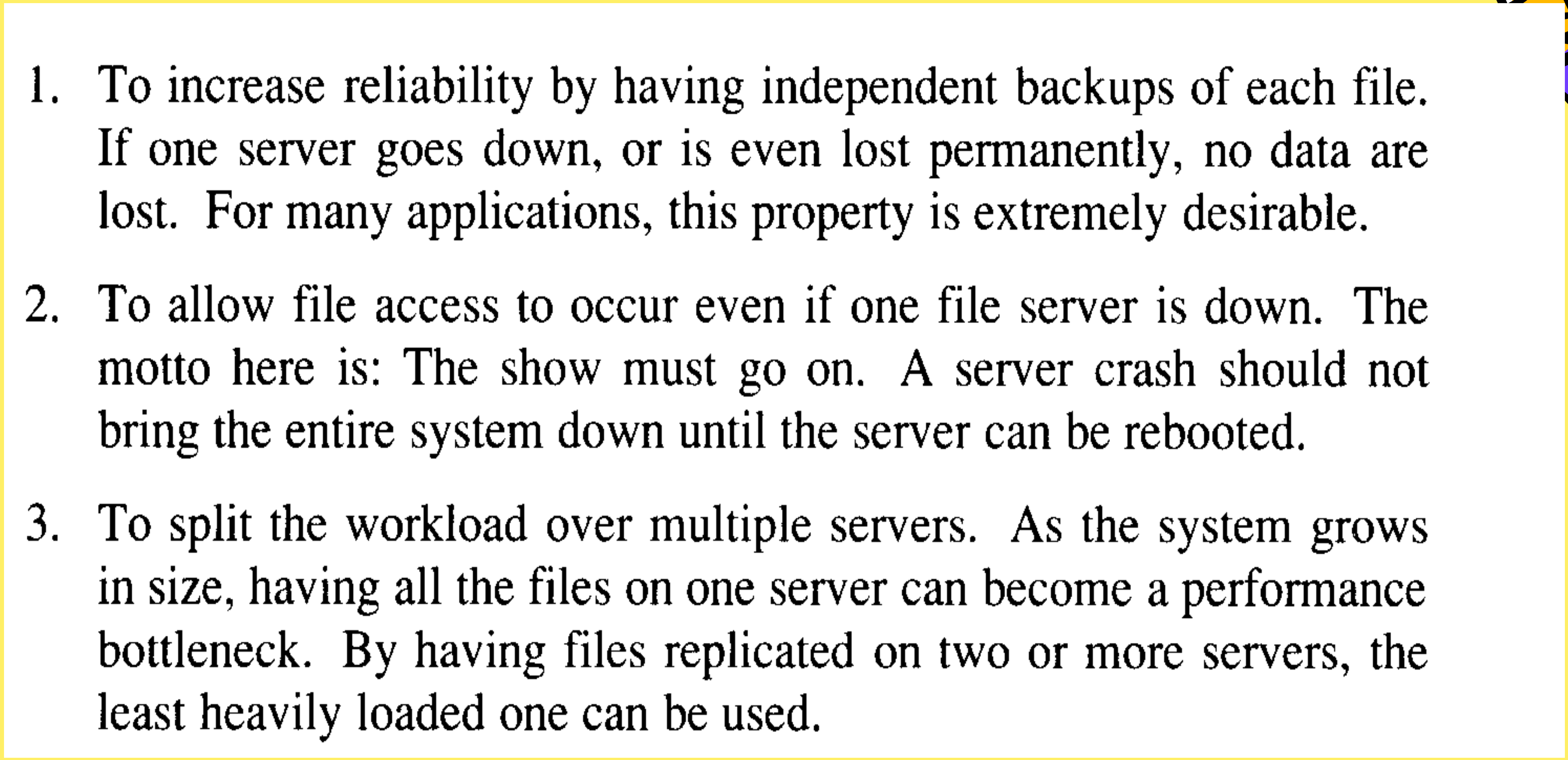
- Server has information about the client between requests.
  - When a Client opens a file (as in Unix & Windows), Server fetches information about the file from its disk, stores it in its memory, and gives the client a unique connection identifier for the open file
  - The server has information about which client has which file open.
  - Identifier is used for subsequent accesses by this client until the session ends
  - Server must reclaim the main-memory space used by clients who are no longer active

- Increased performance
  - Fewer disk accesses
  - Stateful server knows if a file was opened for sequential access and can thus read ahead the next blocks
    - E.g., read ahead next blocks for sequential access

DFS –Server Semantics Comparison

- Failure Recovery:
- *In Stateful server,* it loses all volatile state in a crash.
  - Restore old state about the clients by recovery protocol based on a dialog with clients.
  - Server needs to be aware of crashed client processes
    - orphan detection and elimination.
- *In Stateless server* failure and recovery are almost unnoticeable.
  - Newly restarted server responds to self-contained requests without difficulty.

DFS – Replication

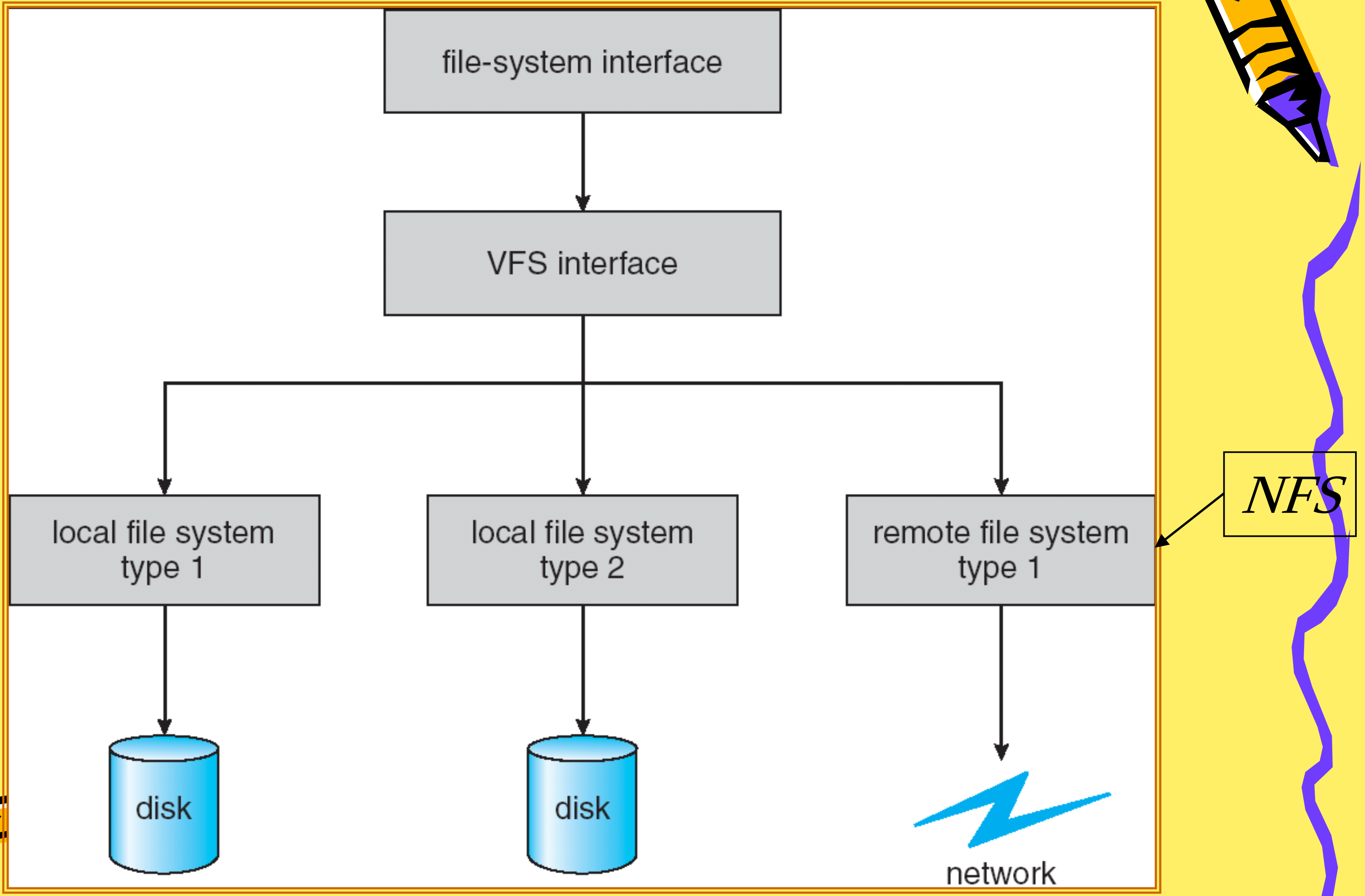


1. To increase reliability by having independent backups of each file. If one server goes down, or is even lost permanently, no data are lost. For many applications, this property is extremely desirable.
2. To allow file access to occur even if one file server is down. The motto here is: The show must go on. A server crash should not bring the entire system down until the server can be rebooted.
3. To split the workload over multiple servers. As the system grows in size, having all the files on one server can become a performance bottleneck. By having files replicated on two or more servers, the least heavily loaded one can be used.

DFS – Replication

- Distributed File systems often provide file replication as a service to their clients.
- Done for selected important files with each copy on different server.
- Advantages: Improves availability (backups are available) and can shorten service time by introducing multiple servers. Replication transparency
- Naming scheme maps a replicated file name to a particular replica.
  - Existence of replicas should be invisible to higher levels.
  - Replicas must be distinguished from one another by different lower-level names.
- Updates
  - Replicas of a file denote the same logical entity
  - Update to any replica *must* be reflected on all other replicas.

NFS Implementation



file-system interface
VFS interface
local file system type 1
local file system type 2
remote file system type 1
NFS
disk
disk
network